Program Synthesis

Discussion

- **Constraint of program syntax**: All of these methods provide the syntax of programs as an inductive bias. A fixed syntax obviously constrains the space of concepts that can be learned. In order to increase the flexibility of learnable concepts, two questions to ask are:
    - Is there a program syntax that can encapsulate the space of all possible rules? (a Turing complete program syntax). Even if such a thing existed, search complexity on it would be intractable.
    - Can program learning be meta learned? By learning a few programs from given data, can you become better at learning programs from another of data set i.e. that follow different grammar? None of the papers test this.
- **Online learning**: Model proposed in [9] needs to be trained offline and applied during user interaction. To make interaction easier, it would help if the model could adapt to the user during interaction i.e. learn online.
- **Analogues**: Symbolic vs neural representations is analogous to System 2 vs System 1 thinking in that the former is interpretable while the latter is not. Also, a major question is regarding the origin of discrete 'concepts'. Neurosymbolic models like [1] provide the program representation as an inductive bias. But in neuroscience, it seems as though instead of discreteness being hardcoded into the architecture, it emerges from neural representations. Similar to original Meta RL paper where meta learning emerges in RNNs learning to perform tasks.

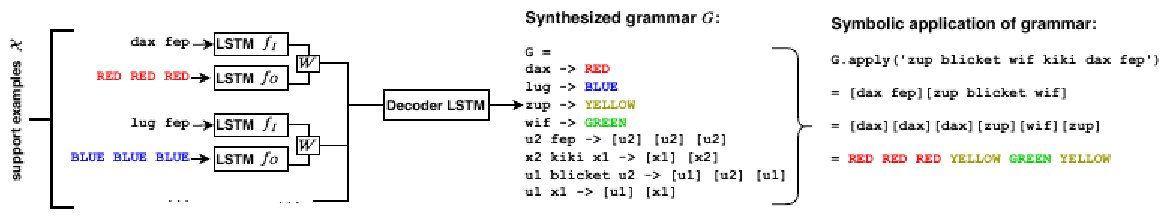
Datasets

1. **MiniSCAN** [5]: The goal of this domain is to learn compositional, language-like rules from a very limited number of examples.
2. **SCAN** [4]: Consists of synthetic language command paired with discrete action sequence. The goal of SCAN is to test the compositional abilities of neural networks when test data varies systematically from training data

Papers

1. Learning Compositional Rules via Neural Program Synthesis

Authors: Nye, Solar-Lezama, Tenenbaum, Lake. NeurIPS 2020.



```
G =
dax -> RED
lug -> BLUE
zup -> YELLOW
wif -> GREEN
u2 fep -> [u2] [u2] [u2]
x2 kiki x1 -> [x1] [x2]
u1 blicket u2 -> [u1] [u2] [u1]
u1 x1 -> [u1] [x1]
```

```
G.apply('zup blicket wif kiki dax fep')

= [dax fep][zup blicket wif]

= [dax][dax][dax][zup][wif][zup]

= RED RED RED YELLOW GREEN YELLOW
```

Figure 2: Illustration of our synthesis-based rule learner neural architecture and grammar application. Support examples are encoded via BiLSTMs. The decoder LSTM attends over the resulting vectors and decodes a grammar, which can be symbolically applied to held out query inputs. Middle: an example of a fully synthesized grammar which solves the task in Figure 3.

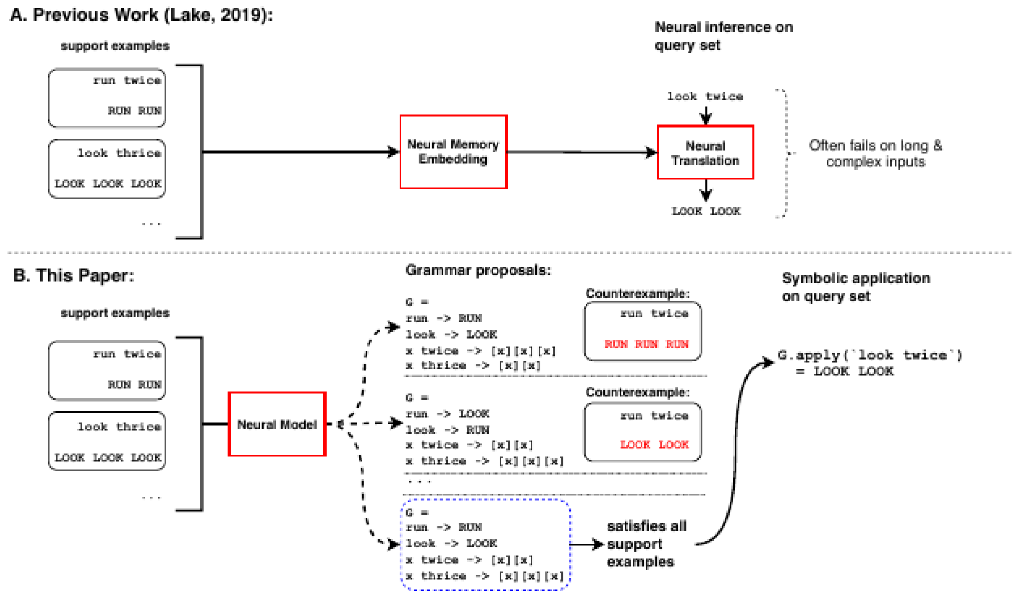

A. Previous Work (Lake, 2019):
support examples
run twice
RUN RUN
look thrice
LOOK LOOK LOOK
Neural Memory Embedding
Neural inference on query set
look twice
Neural Translation
LOOK LOOK
Often fails on long & complex inputs
B. This Paper:
support examples
run twice
RUN RUN
look thrice
LOOK LOOK LOOK
Neural Model
Grammar proposals:
G = run -> RUN look -> LOOK x twice -> [x][x][x] x thrice -> [x][x]
Counterexample: run twice RUN RUN RUN
G = run -> LOOK look -> RUN x twice -> [x][x] x thrice -> [x][x][x]
Counterexample: run twice LOOK LOOK
G = run -> RUN look -> LOOK x twice -> [x][x] x thrice -> [x][x][x]
satisfies all support examples
Symbolic application on query set
G.apply('look twice') = LOOK LOOK

Figure 1: Illustration of our synthesis-based rule learner and comparison to previous work. A) Previous work [9]: Support examples are encoded into an external neural memory. A query output is predicted by conditioning on the query input sequence and interacting with the external memory via attention. B) Our model: Given a support set of input-output examples, our model produces a distribution over candidate grammars. We sample from this distribution, and symbolically check consistency of each sampled grammar against the support set until a grammar is found which satisfies the input-output examples in the support set. This approach allows much more effective search than selecting the maximum likelihood grammar from the network.

- **Task:** Few shot compositional rule learning from training data (support inputs) consisting of query-result pairs.
- **Method:** Generates an explicitly symbolic 'grammar' program using a neural encoder-decoder pair which is run on unseen query inputs to yield results.
  - **Training**: Each episode consists of a single grammar being sampled from the grammar distribution. Compatible support set of 10-20 examples are generated. Using supervised learning, the model is trained to generate the grammar given support examples. Even [2] follows the same training procedure with one difference: since it doesn't use grammars, its trained on a seq2seq problem of generating support output given input.
  - **Testing**: Grammar is sampled from learned synthesis model. A search process involves checking how well the grammar satisfies support examples. This runs until either a grammar satisfies all of them or search budget runs out in which case, grammar which satisfied most no. of examples is chosen. Unlike seq2seq model which is forced to operate

with network learned from training process (i.e. it gets only 1 try), this model can guess-and-check until it gets everything right.

- **Model architecture:** 1 BiLSTM each for support input and output, followed by hidden layer which combines them to form a single vector representation (h_i) for a support example. LSTM decoder outputs sequence token by token while attending to the support input hidden states (h_i) via attention. Note that attention is done between support examples which ensures that this approach can focus on relevant examples at each decoding step (unlike RobustFill [3] which uses separate encoder decoders for each example). Output tokens are parsed into grammar.
- **Results:**

4.1 MiniSCAN

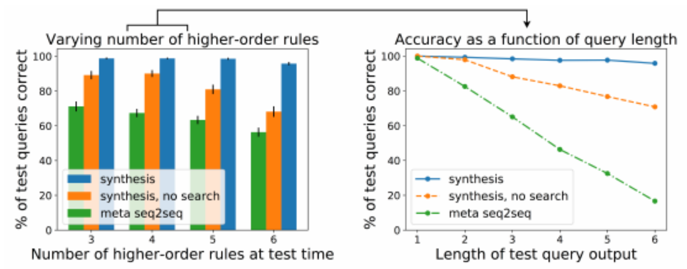


Figure 4: MiniSCAN generalization results. We train on random grammars with 3-4 primitives, 2-4 higher order rules, and 10-20 support examples. Left: At test time, we vary the number of higher-order rules. The synthesis-based approach using search achieves near-perfect accuracy for most test conditions. Right: Length generalization results. A key challenge for compositional learning is generalization across lengths. We plot accuracy as a function of query output length for the "4 higher-order rules" test condition. The accuracy of our synthesis approach does not degrade as a function of query output length, whereas the performance of baselines decreases.

Table 1: Accuracy on SCAN splits.

	length	simple	jump	right
Synth (Ours)	**100**	**100**	**100**	**100**
Synth (no search)	0.0	13.3	3.5	0.0
Meta Seq2Seq	0.04	0.88	0.51	0.03
MCMC	0.02	0.0	0.01	0.01
Sampling from prior	0.04	0.03	0.03	0.01
Enumeration	0.0	0.0	0.0	0.0
DeepCoder	0.0	0.03	0.0	0.0
GECA [14]	–	–	87	82
Meta Seq2Seq (perm)	16.64	–	99.95	98.71
Syntactic attention	15.2	–	78.4	28.9
Seq2Seq [4]	13.8	99.8	0.08	–

Table 2: Accuracy on few-shot number-word learning, using a maximum timeout of 45 seconds.

	English	Spanish	Chinese	Japanese	Italian	Greek	Korean	French	Viet.
Synth (Ours)	**100**	**80.0**	**100**	**100**	**100**	**94.5**	**100**	**75.5**	**69.5**
Synth (no search)	**100**	0.0	**100**	**100**	**100**	70.0	**100**	0.0	**69.5**
Meta Seq2Seq	68.6	64.4	63.6	46.1	73.7	89.0	45.8	40.0	36.6

- Also, expected to be more robust (due to checking mechanism) and interpretable (due to explicit symbolic grammar) than neural models.

2. Program Synthesis with Pragmatic Communication

Authors: Pu, Ellis, Kryven, Tenenbaum, Solar-Lezama. NeurIPS 2020.

- **Task**: Program synthesis is the task of learning a program that generates a given specification. Example: For the spec *Richard Feynman → Mr. Feynman*, a program would be `Mr. + last_word(input)`. But its easy for the given spec to under-specify the intent. Example: The synthesizer might learn to just always print *Mr. Feynman* if just given the above example. To prevent this, previous methods use inductive biases. Example: FlashFill [6] penalises constant strings. [7] uses handcrafted features to make up for failures of [6].

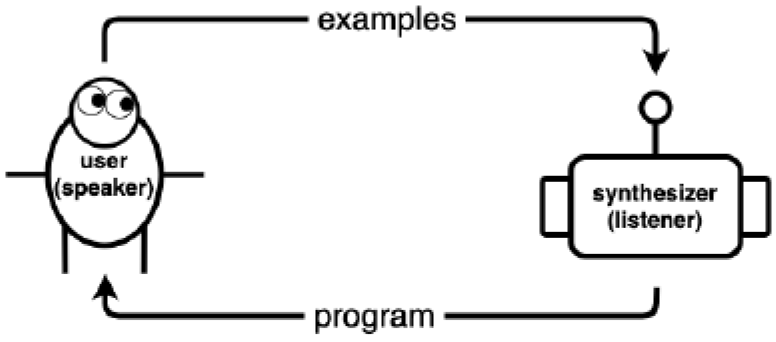


Figure 1: program synthesis as a reference game

- **Formulation as a reference game**: User communicates a concept h (program) using utterances u_i (examples). Synthesizer S's job is to obtain the concept from these utterances. So communication is successful if $h = L(S(h))$ and efficient if the set of utterances is small. Usually, to build in human speaking convention into the model, other approaches use human annotated datasets. But here, the integrating pragmatic communication into the synthesizer enables it to learn without it.
- See paper for proof that resulting convention from speaker listener pair is both efficient and human usable. RL approaches involving communication between agents [8] are good but not human usable.
- **Method:**
    - Literal Listener L_0: Upon receiving a set of examples D, L_0 samples uniformly from the set of consistent concepts.

$$P_{L_0}(h|D) \propto \mathbb{1}(h \vdash D), \quad P_{L_0}(h|D) = \frac{\mathbb{1}(h \vdash D)}{\sum_{h' \in H} \mathbb{1}(h' \vdash D)}$$

- Pragmatic Speaker S_1: Models the speakers generation of examples sequentially.

$$P_{S_1}(D|h) = P_{S_1}(u_1, \ldots, u_k|h) = P_S(u_1|h)P_S(u_2|h, u_1) \ldots P(u_k|h, u_1 \ldots u_{k-1}) \qquad (2)$$

where the incremental probability $P_S(u_i|h, u_1, \ldots, u_{i-1})$ is defined recursively with L_0:

$$P_S(u_i|h, u_{1\ldots i-1}) \propto P_{L_0}(h|u_{1\ldots i}), \quad P_S(u_i|h, u_{1\ldots i-1}) = \frac{P_{L_0}(h|u_1, \ldots, u_i)}{\sum_{u_i'} P_{L_0}(h|u_1, \ldots, u_i')} \qquad (3)$$

- Informative Listener L_1: Reasons about S_1.

$$P_{L_1}(h|D) \propto P_{S_1}(D|h), \quad P_{L_1}(h|D) = \frac{P_{S_1}(D|h)}{\sum_{h'} P_{S_1}(D|h')}$$

This above process is very inefficient. So they propose a more efficient procedure (see paper) using version space algebra and memoization (like DP?). But even after this, its much more inefficient than SOTA program synthesizers.

- **Experiments**: Task is to place some shapes with certain colours on a grid. Space of programs is 17976. Atomic examples are of the form $((x, y), s))$ where first element is coordinates and second is symbol. 343 atomic examples. Human studies were carried out.

```
P   -> if (x,y) in box(B,B,B,B)
       then symbol(S,C)
       else pebble
B   -> 0 | 1 | 2 | 3 | 4 | 5 | 6
S   -> ring(O,I,R,x,y)
O   -> chicken | pig
I   -> chicken | pig | pebble
R   -> 1 | 2 | 3
C   -> [red, green, blue][A2(A1)]
A1  -> x | y | x+y
A2  -> lambda z:0 | lambda z:1 |
       lambda z:2 | lambda z:z%2 |
       lambda z:z%2+1 |
       lambda z:2*(z%2)
```

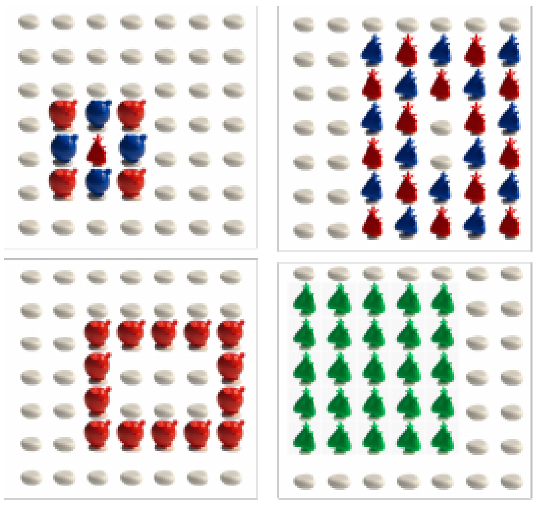


Figure 3: DSL of pattern laying programs / rendering of 4 different programs on 7×7 grids

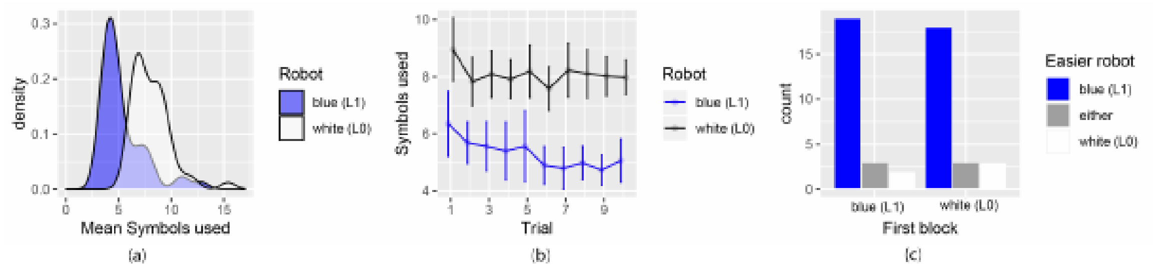
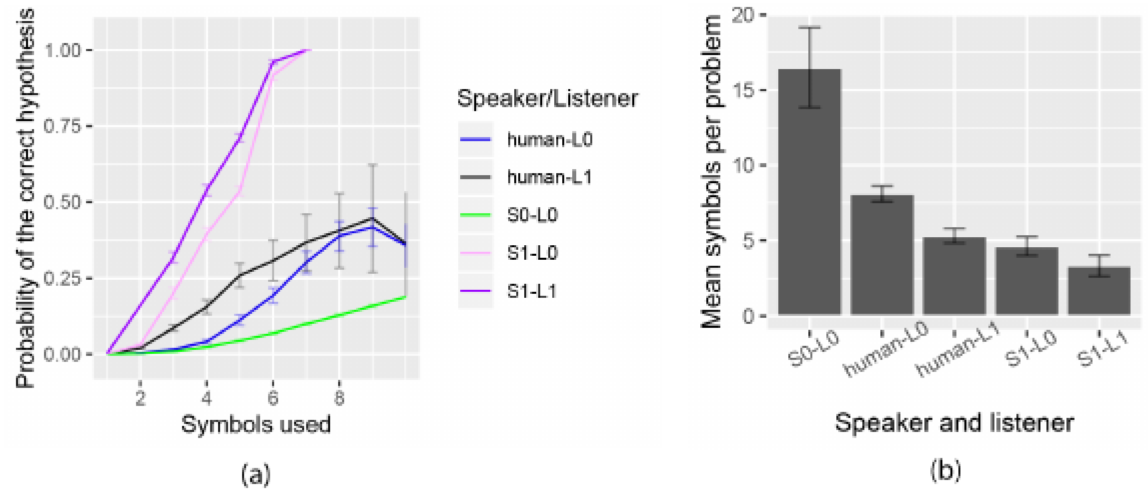


Figure 5: (a) the density of the mean number of symbols used (N=48). (b) the mean number of symbols used by subjects during the course of the experiment (error bars show 95% confidence intervals), communicating with the both robots improvement over time. (c) which robot was easier to communicate with.



3. Learning abstract structure for drawing by efficient motor program induction

Authors: Tian, Ellis, Kryven, Tenenbaum. NeurIPS 2020.

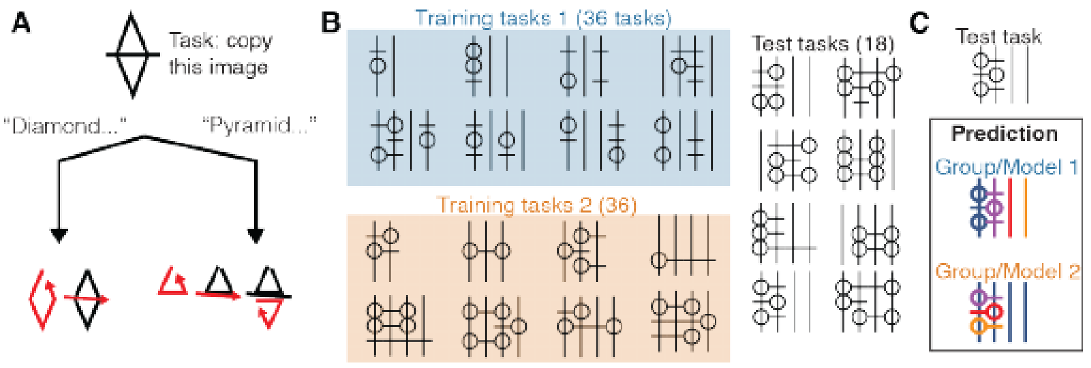


Figure 1: Background and overview. **(A)** Prior knowledge influences how people draw. Consider how one might copy the drawing on top. Copying behavior tends to differ depending on what structural description is associated with the object - "Diamond with a cross-line" vs. "Pyramid and its reflection" [8]. **(B)** Representative tasks in Training sets 1 and 2, and in the common Test set. **(C)** Hypothesized behavior on test tasks that would be diagnostic of learned abstract structure.

- **Task**: To a) understand how humans infer structure from examples, b) develop a program learning based model of this and compare with human subjects on a novel task. Proposes a novel, ambiguous copying task. Model they've developed aims to be *abstract* and *compositional*.
- **Method**:

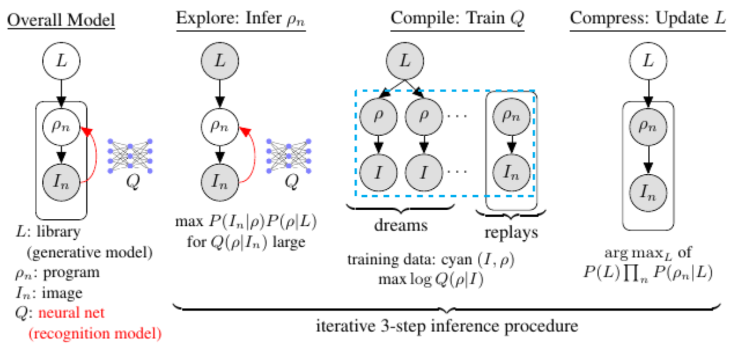

Overall Model
Explore: Infer ρ_n
Compile: Train Q
Compress: Update L
L
ρ_n
I_n
Q
ρ
I
L: library (generative model)
ρ_n: program
I_n: image
Q: neural net (recognition model)
max $P(I_n|\rho)P(\rho|L)$ for $Q(\rho|I_n)$ large
dreams
replays
training data: cyan (I, ρ) max $\log Q(\rho|I)$
arg max$_L$ of $P(L)\prod_n P(\rho_n|L)$
iterative 3-step inference procedure

Figure 2: The Bayesian neurosymbolic program induction algorithm that underlies our computational model, based on [23, 33]. Left ("overall model"): Each observed image I_n is explained using a latent program ρ_n. The prior or inductive bias is modeled by an inventory or "library" of learned primitives, L. A neural network recognition model (red arrows) learns to map from images to a distribution over source code of programs likely to explain that image. Conditional distribution output by the network is notated $Q(\cdot|\cdot)$. Inference iterates through **Explore**, which searches programs ordered under Q and rescores them under true posterior $P(\cdot|L, I_n)$; **Compile**, which trains the neural network Q to search for image-explaining programs, training both on replays of programs from Explore and "dreams," or samples from the learned prior; and **Compress**, which updates the prior by compressing out new compositional code abstractions which are incorporated into the library L.

- **Results:**

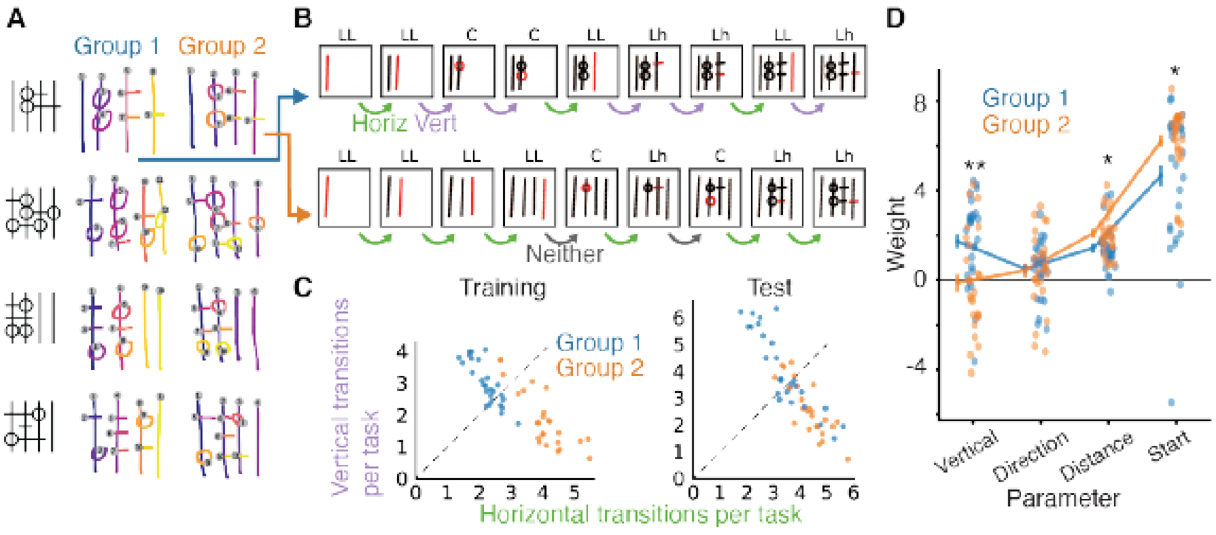
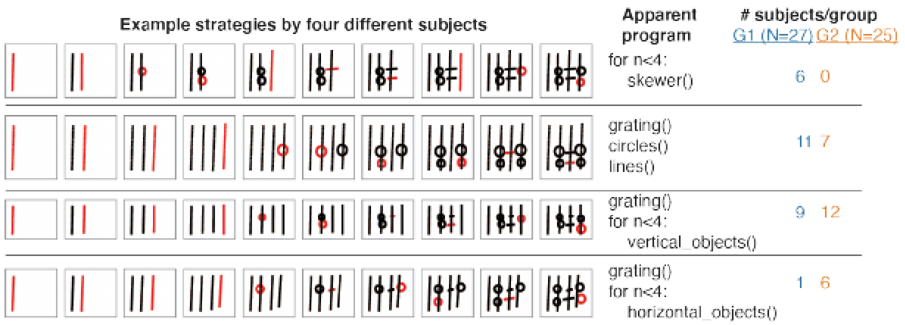


Figure 3: Single-session learning of structure in drawings. **(A)** Example drawing trajectories for two subjects (columns) on four test tasks (rows). Stroke order is indicated by both color (purple to yellow) and the number in grey dot. Grey dots also indicate start positions. **(B)** Example segmentations of behavior into action trajectories. Letter codes indicate stroke categories. **(C)** Stroke transitions were directionally biased for both Training and Test tasks. **(D)** MC feature weights for behavior on Test stimuli. Positive corresponds to a bias for transitions that are vertical, towards bottom-right, low distance, and for first stroke at top-left. *, **, p<.05, .005, t-test.



Example strategies by four different subjects	Apparent program	# subjects/group G1 (N=27)	G2 (N=25)
	for n<4: skewer()	6	0
	grating() circles() lines()	11	7
	grating() for n<4: vertical_objects()	9	12
	grating() for n<4: horizontal_objects()	1	6

Figure 4: Program-like structure in behavior. Four example subjects depicting dominant strategies in our dataset (left), apparent program-like structure (middle), and frequencies of these strategies for the two training groups (see also Suppl. Figure 3).

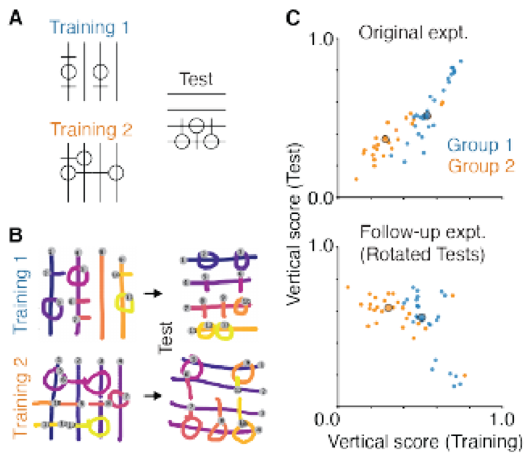


Figure 5: Evidence for abstract generalization in a followup study with rotated test stimuli. **(A)** Test stimuli are rotated relative to the original experiment in Figure 3. **(B)** Example drawings for two subjects showing that they retained program-like biases evident for rotated Test tasks. **(C)** Summary analysis. "Vertical score" is computed as $V/(V+H)$, where V and H are average vertical and horizontal transitions per task. Large dots indicate medians.

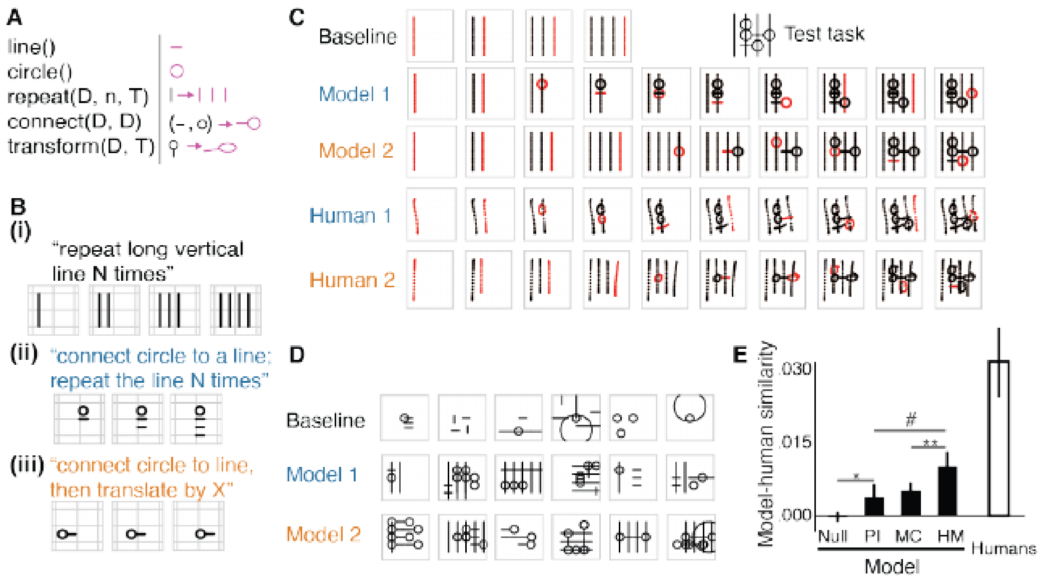


Figure 6: Modeling results. **(A)** Example starting primitives (left) and drawings (right). D, n and T are variables representing drawings, natural number, and transformation (see Table 1). **(B)** Example subroutines learned by both HM1 and HM2 (Bi), HM1 only (Bii), and HM2 only (Biii). **(C)** On an example test task, solutions by models (top) and example humans (bottom). **(D)** Example "dreams," or prior samples at Baseline, and after training on Tasks 1 (HM1) or 2 (HM2).**(E)** Comparing human and model behavior on test tasks. $Similarity = mean(D(H_1, M_2), D(H_2, M_1)) - mean(D(H_1, M_1), D(H_2, M_2))$, where distance $D(H_i, M_j) = \frac{1}{N_h}\frac{1}{N_s}\sum_{h \in H_i}\sum_s d(h, M_j, I_s)$ is distance to model averaged over humans (h) and test stimuli (s), for the group of humans trained on Task set i and the model trained on set j. *, **, p<.05, .005; #, p=.06, paired t-test. Null model p<.05 vs. other models.

References

1. **Nye, Maxwell I., Armando Solar-Lezama, Joshua B. Tenenbaum, and Brenden M. Lake. "Learning Compositional Rules via Neural Program Synthesis." arXiv preprint arXiv:2003.05562 (2020).**

2. Brenden M Lake. Compositional generalization through meta sequence-to-sequence learning.
In Advances in Neural Information Processing Systems, pages 9788–9798, 2019.
3. Amit Zohar and Lior Wolf. Automatic program synthesis of long programs with a learned
garbage collector. In Advances in Neural Information Processing Systems, pages 2094–2103,
2018.
4. Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional
skills of sequence-to-sequence recurrent networks. In 35th International Conference on Machine Learning, ICML 2018, pages 4487–4499. International Machine Learning Society (IMLS), 2018.
5. Brenden M Lake, Tal Linzen, and Marco Baroni. Human few-shot learning of compositional
instructions. Proceedings of the 41st Annual Conference of the Cognitive Science Society, 2019.
6. Oleksandr Polozov and Sumit Gulwani. Flashmeta: A framework for inductive program synthesis. ACM SIGPLAN Notices, 50(10):107–126, 2015.
7. Kevin Ellis and Sumit Gulwani. Learning to learn programs from examples: Going beyond program structure. IJCAI, 2017.
8. Mike Lewis, Denis Yarats, Yann N Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-to-end learning for negotiation dialogues. arXiv preprint arXiv:1706.05125, 2017.
9. **Pu, Yewen, Kevin Ellis, Marta Kryven, Joshua B Tenenbaum, and Armando Solar-Lezama. "Program Synthesis with Pragmatic Communication," n.d., 11.**
10. **Tian, Lucas Y, Kevin Ellis, Marta Kryven, and Joshua B Tenenbaum. "Learning Abstract Structure for Drawing by Efficient Motor Program Induction," n.d., 12.**